

54. IWK
Internationales Wissenschaftliches Kolloquium
International Scientific Colloquium



**Information Technology and Electrical
Engineering - Devices and Systems, Materials
and Technologies for the Future**



Faculty of Electrical Engineering and
Information Technology

Startseite / Index:

<http://www.db-thueringen.de/servlets/DocumentServlet?id=14089>

Impressum

Herausgeber: Der Rektor der Technischen Universität Ilmenau
Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c.
Peter Scharff

Redaktion: Referat Marketing
Andrea Schneider

Fakultät für Elektrotechnik und Informationstechnik
Univ.-Prof. Dr.-Ing. Frank Berger

Redaktionsschluss: 17. August 2009

Technische Realisierung (USB-Flash-Ausgabe):
Institut für Medientechnik an der TU Ilmenau
Dipl.-Ing. Christian Weigel
Dipl.-Ing. Helge Drumm

Technische Realisierung (Online-Ausgabe):
Universitätsbibliothek Ilmenau
[ilmedia](#)
Postfach 10 05 65
98684 Ilmenau

Verlag:  Verlag ISLE, Betriebsstätte des ISLE e.V.
Werner-von-Siemens-Str. 16
98693 Ilmenau

© Technische Universität Ilmenau (Thür.) 2009

Diese Publikationen und alle in ihr enthaltenen Beiträge und Abbildungen sind urheberrechtlich geschützt.

ISBN (USB-Flash-Ausgabe): 978-3-938843-45-1
ISBN (Druckausgabe der Kurzfassungen): 978-3-938843-44-4

Startseite / Index:
<http://www.db-thueringen.de/servlets/DocumentServlet?id=14089>

COMPOUND JOB SCHEDULING AND JOB-FLOWS MANAGEMENT IN DISTRIBUTED COMPUTING

A. Tselishchev, V. V. Toporkov

CERN (European organization for nuclear research);
MPEI (Moscow Power Engineering Institute)

ABSTRACT

The paper presents an approach to the problem of job scheduling management in distributed heterogeneous computational environments with a fixed set of resources. The fact that the architecture of the computational environment is distributed, heterogeneous and dynamic along with the autonomy of processor nodes, makes it difficult to manage and assign resources for job execution at the required quality level.

Index Terms – distributed, computing, scheduling, metascheduler, job-flow, application-level, grid planning, complex structured jobs, collision, resource co-allocation, co-scheduling, planning strategies.

1. INTRODUCTION

Nowadays, there are situations when it is impossible to process large volumes of data in a reasonable amount of time without using distributed computing – the data-flows need to be directed and coordinated in a proper way to process the results of scientific experiments, to conduct computationally intensive research or just to monitor everyday activities on a larger scale. This paper describes the strategies of coordinated resource co-allocation in distributed computing. These strategies are implemented using a combination of job scheduling methods and techniques of application-level scheduling, where applications are regarded as compound jobs with a complex structure. Strategy is considered as a set of possible job scheduling variants on the processor nodes with coordinated planning and assignment of tasks, which are included into a multiprocessor job. The choice of the specific variant depends on the load level of the resource dynamics and is formed as a resource query which is sent to a local batch-job processing system. Safety strategies that take many factors into account (i.e. actual physical availability of resources) and allow implementing different computing scenarios are analyzed.

In the wide range of different approaches to computing management in distributed environments, one can find two polar and settled trends. The first is

based on the usage of the available resources, where resource brokers are acting as agents between users and processor nodes. Several projects such as AppLeS (Application-Level-Scheduling, [1]), APST [2], Legion [3], GC-DRM [4], Condor-G [5], Nimrod-G [6] and others follow this idea and are often associated with application-level scheduling, where no regulations for resource allocation are provided. Another trend is based on the concept of virtual organizations and is mainly aimed at Grid systems. Both trends have their own advantages and disadvantages. Resource brokers, that are used in the first trend are both scalable and flexible and can be adapted to a specific application. On the other hand resource distribution dedicated to a specific application as well as the usage of different criteria by independent users for the respective job execution plan optimization may deteriorate such integral characteristics as completion time for the batch-job or resource load level. This is happening especially while considering possible competition with other jobs.

Distinct from existing solutions, our approach to computing in heterogeneous environments contains mechanisms of dynamic redistribution of job-flow between processor nodes in conjunction with application-level scheduling. It is considered that the job can be compound and the tasks included in the job are heterogeneous in terms of computation volume and resource need. In order to complete the job, one would have to co-allocate the tasks on different processor nodes. Each task is executed on a single processor node and it is supposed that the local management system interprets it as a job that is formed as a resource-query. Fundamental proposal of this technique is that the resultant dispatching strategies are based on the integration of job-flows management methods and compound job scheduling methods on processor nodes. It allows increasing the quality of service for the jobs and distributed environment resource usage efficiency. This paper is related to the description of the basics of practical implementation and simulation of the scheduling processes proposed. The underlying theory is described in [7] and [8].

2. AREA OF APPLICATION AND DEFINITIONS

The nature of computational environments themselves demands the development of multi-criteria and multi-factor strategies of coordinated scheduling and resource allocation. Factors such as the dynamic configuration of the environment, large number of resource reallocation events, users' and resource owners' needs and virtual organization policy of resource assignment should be taken into account. It is also very important to consider the approach proposed not only for the distributed computing area, but for any other activity that involves planning. For example, publishing books, assembling cars or making movies are processes that share similar abstract entities as scheduling of interrelated tasks in distributed computing. One can relate a computational unit (resource) to an assembly worker or illustration designer and computational tasks to a work of any other type. Strategy, being a set of schedules, shows the most effective (according to the chosen strategy type) sequence of actions which leads to the defined goal.

In our case, the goal is to execute and compute a set of computational tasks in a distributed environment, though the approach can surely be implemented on a more general scale. Authors of the paper are researching Grid planning mechanisms, so the terminology used here will be specifically related to this area as follows. *CPU* is defined as an abstract resource, which can be used for execution of an abstract *task*. The complex set of connected interrelated tasks form a *job*. In some applications jobs require co-scheduling [9] and resource co-allocation [10] on several CPUs. In this case resource allocation has a number of substantial specific features caused by autonomy, heterogeneity, dynamic changing of the contents, and failures of nodes [11].

One of the most popular techniques used in distributed planning is pre-emptive scheduling based on queues. This is actually not an efficient way of multiprocessor jobs co-allocating in our opinion. Besides, there are several well-known downsides of this method in the cluster systems such as LL, NQE, LSF, PBS and others. For example, traditional First-Come-First-Serve strategy leads to idle standing resources. Another strategy, which involves job ranking according to the specific properties (such as computational complexity, for example LWF or Most-Significant-First) leads to a severe resource fragmentation, and often makes it impossible to execute some jobs due to the absence of idle resources. In distributed environments these effects can lead to unpredictable job execution time and thereby to unsatisfactory quality of service. In order to avoid this many projects have components which make schedules that are supported by preliminary

resource reservation mechanisms. One example system is the Maui cluster scheduler, where a backfilling algorithm is implemented. The remote Grid resource reservation mechanism is also supported in GARA, Ursala and Silver projects. In these only one variant of the final schedule is built and it can become irrelevant because of changes in the local job-queue, transporting delays etc.

The significant difference between the approach proposed in this paper and well-known scheduling solutions for distributed environments such as the Grid is the fact that the whole set of scheduling co-allocation variants actually forms a strategy. This strategy is formed on a basis of formalized efficiency criteria, which efficiently allow to reflect economical principles of resource allocation by using relevant cost functions, and solving a load balance problem for heterogeneous processor nodes.

3. SIMULATION ENVIRONMENT

Strategy of a compound job execution is a set of possible resource allocation and schedules for each task in the job:

$$Distribution = \langle \langle Task_{\{1\}} / Allocation_{\{i\}}, [Start_{\{1\}}, End_{\{1\}}] \rangle, \dots, \langle Task_{\{N\}} / Allocation_{\{j\}}, [Start_{\{N\}}, End_{\{N\}}] \rangle \rangle,$$

where $Allocation_{\{i,j\}}$ are resources (CPUs) and $Start_N, End_N$ are run time and stop time for task N execution. Time interval $[Start, End]$ is treated as a so called *walltime*, defined at the resource reservation time in the local batch-job processing system. The metascheduler task is to distribute job-flows between processor node domains according to the selected co-allocation strategy. It doesn't mean that these flows cannot intersect each other on processor nodes. A special job-between-processor-nodes reallocation mechanism is provided. It is executed on the higher-level manager or on the metascheduler. Job-managers are supporting and updating strategies based on cooperation with local managers (batch-job processing systems) and simulation approach for job execution on local processor nodes. The authors have implemented a simulation environment for the metascheduler project and made a pilot study of the efficiency indices of different co-allocation strategies, based on different job structure and environment data distribution.

The first version of the software provided the very basic functions for creating strategies for an abstract job generated randomly. Every job is represented as an oriented weighted acyclic graph with vertices corresponding to the tasks and edges corresponding to the precedence relations between tasks. Weights of vertices are proportional to the computational volumes of tasks while weights of edges correspond to data transfer times. The complexity of the graphs

is defined by the following parameters: the number of layers, the maximum number of vertices for one layer and the edge density. Every graph is then processed in three major steps:

1. Forming and ranging a set of critical paths (longest sets of connected tasks) in the graph
2. Consequential planning of every critical path in the graph
3. Resolution of possible collisions

These steps implement the *critical jobs method*, which was earlier developed by the authors [12].

3.1. Critical jobs method – ranging

In order to form a sorted critical paths array for the graph that will include every vertex and every edge only once, the following algorithm is proposed:

- 1) Build the array containing every critical path in the graph by building a tree of critical paths (each vertex in the tree represents one path)
 - a) Calculate the root vertex of the tree (longest critical path P_n for the initial graph $G_{n,1}=\{U,V\}$) with Ford's algorithm [13], current layer $n=0$.
 - b) $n=n+1$
 - c) Calculate critical paths that correspond to the vertices of the current layer. Each vertex corresponds to the graph of the adjacent vertex on $n-1^{th}$ layer, where one edge is taken away. For example if the root critical path P_0 had 5 edges e_1-e_5 from the initial graph, layer $n=1$ will have 5 vertices, that represent 5 graphs $G_{1,1}-G_{1,5}$ and each of them will correspond to the initial graph without one of the five edges. $G_{1,1}=\{U\setminus e_1,V\}$, $G_{1,2}=\{U\setminus e_2,V\}$ etc. Graphs $G_{1,1}-G_{1,5}$ will not contain the path P_0 , that was calculated in the previous layer, so Ford's algorithm will find another critical path for each of those graphs.
 - d) If a vertex corresponds to the graph without edges, it is considered as a leaf
 - e) If each vertex of the current layer is a leaf, then exit the cycle. Otherwise go to b.
- 2) Sort the array of paths by their lengths in descending order (each path corresponds to the vertex of a tree from the step 1)
- 3) Include paths in the final set from the top of the array until this set comprises all vertices and edges of the graph.

3.2. Critical jobs method – planning

The second step begins when critical path array is received by the main planning process and the assignment of computing resources is ready to go. The scheduler expects an oriented graph which is preliminarily split into critical paths as input. Output

of the scheduler is a set of vectors of *a priori* time estimates for each of vertices of the graph which is based on single-criterion optimization (in the general case, it can be multi-criteria but current version of software implements single-criterion optimization). This set was earlier defined as a strategy for a job. Input data requirements consist in following: critical jobs must be ranged i.e. ordered by *a priori* execution time estimates. Execution time estimate is counted as a sum of each task execution time using the fastest processor possible plus sum of the data exchange times between the tasks. Critical paths are defined by tasks belonging to a job and information links conveying the precedence relations. Input data also includes processor performance values for each of the tasks.

The scheduler algorithm contains a cycle which iterates over each critical path. As a result of single iteration one or more distribution variants are produced. These distribution variants are sub-vectors of the final distribution vector that is the output of the whole planning process.

During each iteration, a sub-vector includes *a priori* time estimation for corresponding tasks of a critical job. If there is more than one variant at the start of iteration, then each of them is processed. During this process several new variants may be added to the list, but they can be checked during the next iteration for next critical job.

The cycle includes following steps:

1. If the undistributed task count of the current critical job is greater than 1, a search-tree is created according to the method described in [12], and the optimization is applied. After filling up the search-tree, «reverse» and «forward walk» are conducted that result in one or more sub-vectors as one or more optimums of criteria function could be found during the «forward walk». If a tree has been made for the first critical job, the sub-vectors are copied to final distribution variant list. Otherwise the sub-vectors are merged with existing ones from the list. If a tree has been constructed for other critical job and m variants exist at the moment, then adding up n new variants will produce $m*n$ resulting sub-vectors.

All the tasks that belong to the current critical job are considered as distributed at this point. There is a possibility of search-tree processing failure which can be the result of a too-tight time (or any other variable) constraint. If this is the case the current variant is considered invalid and is removed from the list later.

2. If a critical job contains only one undistributed task, its time boundaries are specified within the already distributed tasks of the job. If there is a processor that can run the undistributed task in the specified time, then *a priori* time estimation for this task and processor is saved to the variant, otherwise the current variant is considered invalid. The variant

count can decrease or remain the same during this type of iteration.

3. If a critical job contains only one information link (edge in the graph), which can happen if a job consists of two tasks and both are distributed to the moment of iteration start, a simple check of time boundaries is performed. If the data exchange provided with information link does not fit into the time boundaries that are specified by the distributed tasks (which may happen if their distributions overlap), then the current variant is also considered invalid.

4. If a critical job contains the only isolated task, then the whole planning scale (time reserve) becomes its time execution estimate. Such an assignment depends on criteria used for planning which is average processor utilization. With other criteria functions the assignment rule for an isolated task job may vary. The variant count does not change during this type of iteration.

5. When all tasks of all critical jobs are distributed or the variant list is empty the planning process is finished. As a result, the ranged set of optimal schedules is forwarded to the possible collisions resolution module. This schedule is called the *preliminary schedule*.

3.3. Critical jobs method – collisions

The planning step of the critical jobs method considers the types of CPUs to which tasks can be assigned. It is assumed that there is an unlimited number of each CPU type for task assignment, though the method can easily be adapted for processing real CPU entities and not the CPU types if the preliminary schedule is then processed by the collision resolution module (which is also implemented in the simulation environment).

The collision resolution module checks for possible intersections of the assigned tasks in the preliminary schedule (i.e. if two or more tasks are assigned on the same CPU during the same time interval) and reassigns tasks to other CPUs according to the defined policy. Algorithms and heuristics proposed for this reassignment include bipartite graph analysis, where every reassignment is efficiently calculated and leads to the minimal value of the defined penalty criterion. The final list of schedules is considered to be the *strategy* for the planning of a given job.

4. SIMULATION RESULTS

The program model allows setting all parameters of the random job-flow, schedule criteria, penalty functions, and the frequency of jobs submission. After the simulation timelines of resource utilization and detailed information about corresponding collisions and the way they were resolved is shown.

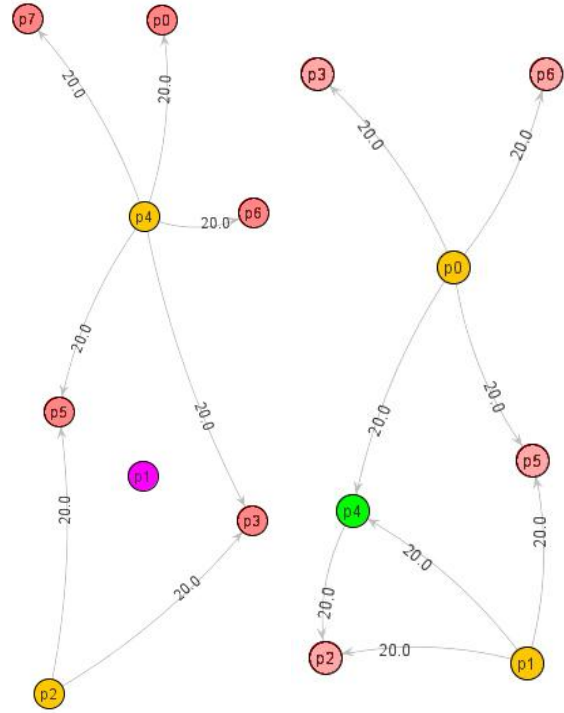


Figure 1 Sample jobs

After successful testing of the developed algorithms the revision of all operational parts of the scheduler and preparation of the kernel for the integration into a real environment was considered. Demonstration and operational modules were separated and optimized. A full refactoring and profiling of the project source code led to a performance increase of 400%. Simple example describing the result of processing two jobs by the system is explained below. Fig. 1 represents two jobs with walltimes $t_1=110$ and $t_2=140$ that are submitted to the distributed environment with 8 abstract CPUs. If the jobs are submitted one-by-one the metascheduler will also schedule them one-by-one and will guarantee that every job will be scheduled within the defined time interval and in a most efficient way in terms of a selected penalty cost-function and maximize average load balance of CPUs on a single job scale (Fig. 2). Job-flow execution will be finished at $t_3=250$. This is a classic example of application-level scheduling and no integral job-flow characteristics are optimized in this case.

To combine application-level scheduling and job-flow-level scheduling and to fully exploit the advantages of the approach proposed, one can submit both jobs simultaneously or store them in a buffer and execute the scheduling for all jobs in the buffer after a certain amount of time (buffer time). If the metascheduler gets more than one job to schedule it runs the developed mechanisms that optimize the whole job-flow (two jobs in this example). In that case the metascheduler will still try to find an optimal schedule for each single job as described

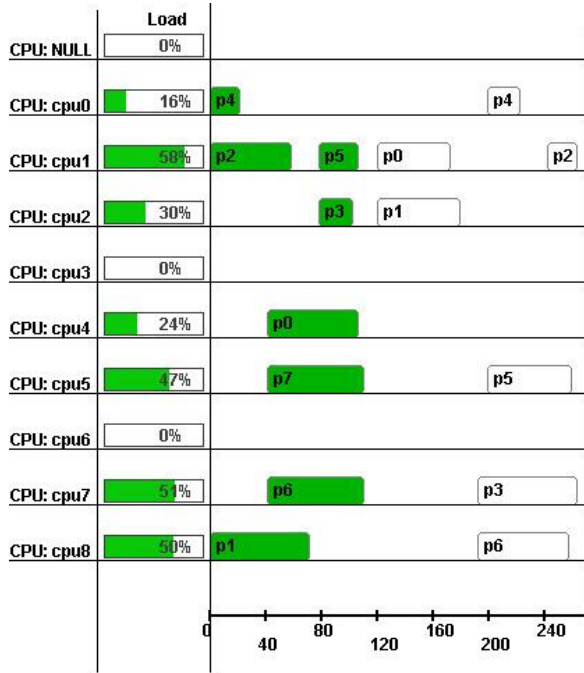


Figure 2 Consequential scheduling

above and, at the same time, will try to find the most optimal job assignment so that the average load of CPUs will be maximized on a job-flow scale. Fig. 3 shows, that both jobs are executed within $t_4=t_2=140$, every data dependency is taken into account (e.g. for the second job: task p_2 is executed only after tasks p_0, p_4 and p_1 are ready), and the final schedule is chosen from the calculated strategy with the lowest penalty function value.

It is important to mention that users can submit jobs without information about the task execution order as required by existing schedulers like the Maui cluster scheduler [14] where only queues are supported. Implemented mechanisms of our approach support a complex structure for the job, which is represented as a directed graph, so users should only provide data dependencies between tasks (i.e. the structure of a job) and the metascheduler will calculate the schedules to satisfy their needs by providing optimal plans for the jobs (application-level scheduling) and the needs for the resource owners by optimizing the defined characteristics of the job-flow for the distributed system (job-flow scheduling).

The last version of the simulation environment was implemented as a two-module system: the first generates jobs and the second makes schedules. Jobs are submitted in the XML format with the description of all data dependencies between the connected tasks. Settings for the generating module allow defining parameters of the job-flow and settings for the metascheduling module allow defining the policy for the scheduling. Currently, two criteria functions can be chosen for application-level scheduling (lowest cost of the schedule and maximum average CPU utilization for one single job) and one for the job-flow scheduling (maximum average CPU utilization for

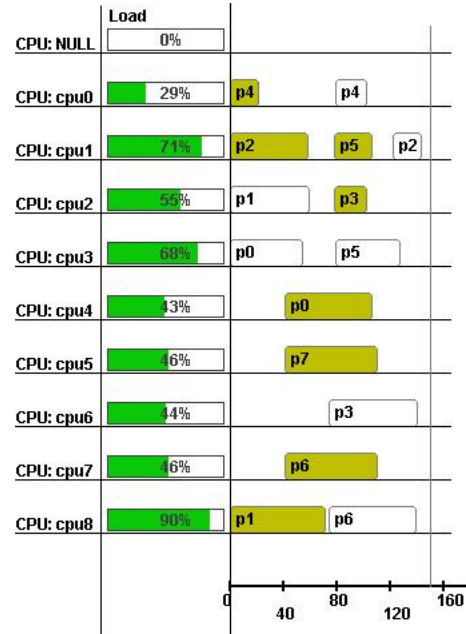


Figure 3 Collective scheduling

the job-flow). Planning module gathers the statistics of the planning, including the overall cost (in terms of the defined penalty function) of every job execution and the whole job-flow, average load balance of CPUs and others.

In order to combine application-level and job-flow scheduling the *critical jobs method* was improved and now operates with a dynamic set of CPUs, checking the availability of each CPU every time an assignment or reassignment is made. It guarantees the optimal schedule for the single job and the metascheduling module allocates only idle CPUs for task assignment, so possible job intersections are avoided and the overall characteristics of the job-flow are optimized. Currently only the basic first-come-first-served (FCFS) management policy is implemented in the metascheduler module, so every job or a set of jobs as described in the example is planned and assigned at the moment it comes to the system, but as the project kernel was reorganized it is now possible to implement more advanced techniques such as gang-scheduling or buffer-based planning.

5. CONCLUSION

In this paper, the practical implementation basics of the combined approach proposed for the resource co-allocation in distributed computing systems are presented. Authors plan to implement advanced scheduling policies in the project to make the metascheduler more flexible. Further improvements of the simulation system will also include the implementation of different criteria functions for job and job-flow scheduling.

At present, when the importance of effective resource utilization is very high in the different areas, it is

obvious that the successful completion of the project will make it possible to create a powerful tool that will allow the execution of complex structured distributed scenarios with complex data dependencies on a set of resources in an optimal way.

6. ACKNOWLEDGEMENTS

This work is supported by the Russian Foundation for Basic Research, project no. 09-01-00095.

7. REFERENCES

- [1] Adaptive computing on the Grid using AppLeS Berman, F.; Wolski, R.; Casanova, H.; Cirne, W.; Dail, H.; Faerman, M.; Figueira, S.; Hayes, J.; Obertelli, G.; Schopf, J.; Shao, G.; Smallen, S.; Spring, N.; Su, A.; Zagorodnov, D. Parallel and Distributed Systems, IEEE Transactions on Volume 14, Issue 4, April 2003 Page(s): 369 – 382 Digital Object Identifier 10.1109/TPDS.2003.1195409
- [2] Practical Divisible Load Scheduling on Grid Platforms with APST-DV van der Raadt, K.; Yang Yang; Casanova, H. Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International Volume , Issue , 04-08 April 2005 Page(s): 29b - 29b Digital Object Identifier 10.1109/IPDPS.2005.351
- [3] Michael J. Lewis, Adam J. Ferrari, Marty A. Humphrey, John F. Karpovich, Mark M. Morgan, Anand Natrajan, Anh Nguyen-Tuong, Glenn S. Wasson and Andrew S. Grimshaw, "Support for Extensibility and Site Autonomy in the Legion Grid System Object Model," Journal of Parallel and Distributed Computing, Volume 63, Number 5, pp. 525-38, May 2003.
- [4] Digital right management based on Grid Computing architecture (GC-DRM) Min-Jen Tsai; Yuan-Fu Luo Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on Volume , Issue , 16-18 April 2008 Page(s):494 – 500 Digital Object Identifier 10.1109/CSCWD.2008.4537028
- [5] <http://www.cs.wisc.edu/condor/condorg/>
- [6] <http://www.csse.monash.edu.au/~davida/nimrod/nimrodg.htm>
- [7] Toporkov, V.V. Tselishchev, A. Safety Strategies of Scheduling and Resource Co-allocation in Distributed Computing, Proceedings of the 2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX ISBN:978-0-7695-3179-3
- [8] V. V. Toporkov Supporting Schedules of Resource Co-Allocation for Distributed Computing in Scalable Systems ISSN 0361-7688, Programming and Computer Software, 2008, Vol. 34, No. 3, pp. 160–172. © Pleiades Publishing, Ltd., 2008.Original Russian Text V.V. Toporkov, 2008, published in Programmirovaniye, 2008, Vol. 34, No. 3.
- [9] M.A. Ioannidou and H.D. Karatza, "Multi-site Scheduling with Multiple Job Reservations and Forecasting Methods", Proc. of the ISPA 2006, LNCS, Vol. 4330, Springer-Verlag Berlin Heidelberg, 2006, pp. 894-903.
- [10] K. Kurowski, J.Nabrzyski, A. Oleksiak, et. al, "Multicriteria Aspects of Grid Resource Management". In: J. Nabrzyski, J.M. Schopf, J. Weglarz (eds), Grid Resource Management. State of the Art and Future Trends, Kluwer Acad. Publ., 2003, pp. 271-293.
- [11] V. Voevodin, "The Solution of Large Problems in Distributed Computational Media",Automation and Remote Control, Pleiades Publishing, Inc., 2007, Vol. 68, No. 5, pp.773-786.
- [12] V.V. Toporkov Modeli raspredelennykh vychislenii (Models of Distributed Computing), Phisimatlit 2004, Moscow, 320p ISBN:5-9221-0495-0 (in Russian)
- [13] Lester R. Ford jr., D. R. Fulkerson: Flows in Networks, Princeton University Press, 1962
- [14] <http://www.clusterresources.com/>